Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Author: Oliver Butler (ob3g21)

Date: 12 January 2024

COMP3222 Machine Learning Technologies Coursework

Analysing and developing a fake information detector using the MediaEval2015 dataset and machine learning

1. Introduction and Data Analysis

1.1 The Task and Problem

Social media is riddled with misinformation. When high-impact new events around significant and controversial topics such as natural disasters, political parties or everyday life occur, it is important to only share correct information. Many news professionals or just normal people who rely on using user-generated media to gain more insight can be misled; therefore they need to easily be able to tell what is real or fake.

The task at hand is to then develop two pipelines for classifying real or fake posts. We take the definition of a fake post to be:

- Reposting of real multimedia, such as real photos from the past re-posted as being associated with a current event, or
- Digitally manipulated multimedia, or
- Synthetic multimedia, such as artworks or snapshots presented as real imagery

However, we shall only use the text for classifying, so any videos, images or audio which can be linked to a topic will not be used. Furthermore, only one dataset, MediaEval2015, will be used for training and testing.

1.2 The Dataset

The MediaEval2015 dataset, the one used in this project, comprises a tab-separated file of multimedia posts with different classifications. A subset of the dataset was used to split into training and testing files. The data consisted of:

- **tweetId**: A non-null int64 unique* ID for each tweet
- **tweetText**: A non-null String consisting of text, emojis, uncleaned data and URLs
- **usedId**: A non-null int64 ID which matched a person's username
- **imageId(s)**: A non-null comma-separated String which links the tweet to an image
- **username**: A non-null String for the username of the person's tweet
- **timestamp**: A non-null Timestamp in the format "WeekDay Month Day HH:MM:SS +0000 Year"
- **label**: A non-null String either "real", "fake" or "humor" ("real or "fake" in test)

1.3 Data Anomalies and Formatting Issues

Both the training and testing datasets have some anomalies regarding the data supplied, where some values are incorrectly formatted. Firstly, in the training dataset, a tweetId should be unique per text. However, in the training dataset, two tweets have the same ID. "263351427320131584", which has the same tweetText but different labels, and "264736470089216000", which has the same values for all categories.

The last 23 rows of the training dataset also have incorrect formatting for both the timestamps and any links in the tweetText columns. The HH:MM:SS part of the timestamp gets formatted to

HH: MM: SS. Links, which should be in the format http(s)://t.co/(link), are in the format http(s):\/\/t.co\/(link) likely due to formatting issues causing the escape character '/' to be added.

Labels in the training dataset should also be converted to only "real" or "fake", where "humor" should be translated to "fake". This is because we should only create a binary classifier for detecting fake text. Whilst you can try to use the "humor" label likely to create a multi-class classifier and convert anything that is labelled as "humor" to "fake", it was deemed unnecessary to do so as binary classifiers tend to be less complicated and have better accuracy rates.

Lastly, tweets should only be in the range of 1-140 characters as at the time of the dataset, tweets were limited to 140 characters. However, both the training and testing datasets have tweets over the limit due to formatting issues. Likely because the text was not formatted to be UTF8 compliant. This caused some emojis to appear incorrectly or '&' being represented as "&". Also incorrectly formatted links increased the length. For example, the longest tweet was 237 characters (Appendix A). Some tweets also heavily consist of spelling mistakes and emojis, which could make them harder to classify, with the worst being shown in (Appendix B)

1.4 Exploring the Dataset

The training dataset has 14483 rows and the testing has 3781 giving an 80:20 split overall. The training dataset consists of 5009 real and 9474 fake labels, meaning the data is skewed towards having mainly fake data which could make detecting real information more difficult. The test dataset has 1217 real and 2564 fake.

Each tweet has some topic/event that surrounds the meaning of the tweet. This can be found using the imageId(s) column. A full list can be found in (Appendix C). However, as there is no overlap between both datasets, it is unlikely that then using this imageId(s) column would provide much use.

The username/userId column shows that 7 people appear in both the training and test set. This could marginally improve the identification of real or fake tweets if we assumed that they posted the tweets they posted were the same label for all of them. However, 5 of the 7 posted tweets with different labels meaning it would be unlikely that these columns would also provide much use.

The timestamp column shows that the training dataset was collected with tweets from October 2012 to September 2014. The test dataset had data from November 2014 to May 2015. This shows why the train and test tweets didn't have similar events. Therefore, like the imageId(s) column, the full timestamp of a tweet wouldn't be correlated with a real or fake value.

The tweetText column consists of many different languages. The most common language was English with a full list found in (Appendix D). Some languages were also more likely to post real or fake information. For example of the 629 Somali tweets, 529 were fake. Seeing how there is double the amount of fake information than real, it could be useful to use the languages to an advantage in detecting fake news and whether translating it has any effect on the performance. A full list of the real and fake languages can be seen in (Appendix E). It is also worth mentioning that the library used to detect the language is non-deterministic and can give different results for the same data given. Seeing how tweets consist of links, emojis, spelling mistakes and hashtags, it could be likely that the languages detected may vary.

Generating a word cloud also gave some insight into different words and events that could help with detecting whether or not text is real or fake. These can be found in Figures 1.4.1 and 1.4.2. However, as the data has yet to be cleaned some words did not appear correctly formatted and the link domain was the most commonly found word. Also as most of the text is in English, other languages did not appear. It is also apparent that links are the most commonly said thing in either dataset, making it unlikely to hold much use. Emojis on the other hand seem to appear much more often in the fake dataset, so they should be explored.

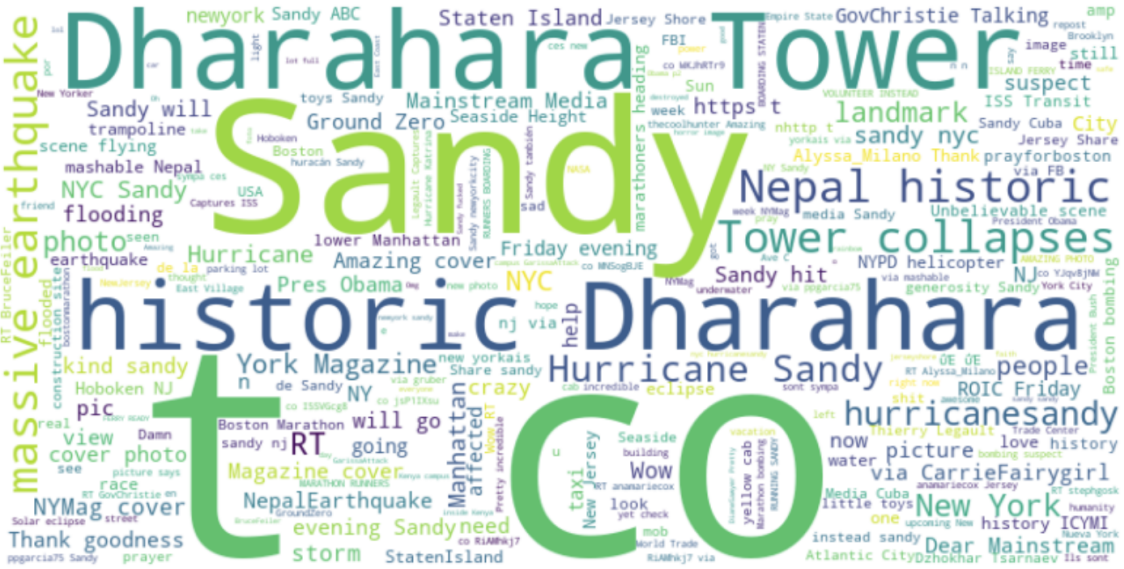

Figure 1.4.1. Generated Word Cloud of the Train and Test dataset with the “real” label limited to 250 words

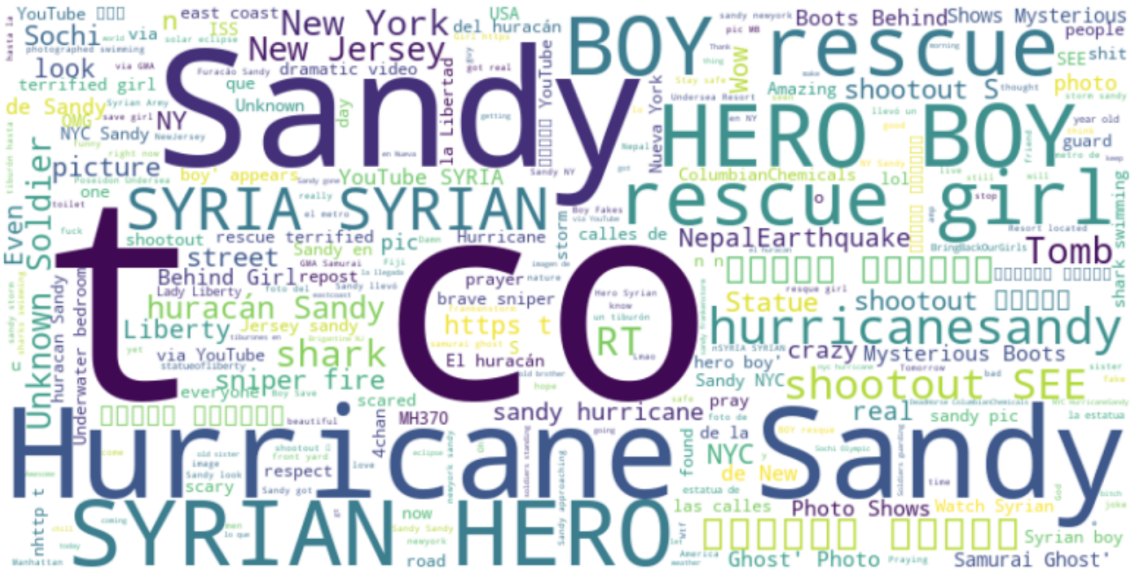

Figure 1.4.2 Generated Word Cloud of the Train and Test dataset with the “fake” label

2. Pipeline Design

2.1 Preprocessing and Fixing Formatting Issues

As mentioned before, the training dataset should have all “humor” labels converted to “fake”. I also chose to change both the “fake” and “real” labels to 1 and 0 respectively due to issues with setting threshold limits for the machine learning models. The original names will still be used though for simplicity's sake in this report onwards.

Next, to fix the formatting issues, all “&” was converted to “&” and removed all emojis from the text. A new column named “emoji(s)” was created by translating any emoji into its text format to see whether or not the emojis were useful in classifying text. Without translating the emojis into text, they were unable to be detected by the models and did not have any effect.

All links were also converted to be in the proper format. A new column named “link(s)” was also created to store the subdirectory of all links to be used later. Then all links were removed from the cleaned tweet text. The reasoning behind this was the links led to some multimedia (image, video or audio) which were mostly inaccessible as they were mostly dead links. It was also told that we were not supposed to use that information in any way. Some links were also repeated multiple times in both the training and testing, but neither had any link which was the same. The link(s) column was only created to test this hypothesis.

This meant all text was now in the 1-140 characters range. It is also worth mentioning the timestamps were not fixed as they were unused in any part of the pipeline due to having no significance in determining a higher accuracy.

To hopefully improve the heavy bias towards English text, using Googletrans, a library which uses an API to communicate with Google Translate, all text was converted to English. This was a tedious process that took 2-3+ days as there were rate limits imposed. This translated most of the text to English, roughly 16472/18264, but some were left in their original language. However, because both the Langdetect and Googletrans libraries are non-deterministic, people who replicated this stage could likely get different results with the detected language and returned text. This was stored then in a new column to test the differences in translating text.

Lastly, all non-alphanumeric characters were removed from the text to help with removing any random character that could affect the next few stages. Then, all words were both stemmed and lemmatised. Then all stop words were removed to help lower the dimensionality of the dataset. This process was improved once most text was translated into English, as most libraries are primarily made to work with the English language.

2.2 Choosing Two Pipelines

2.2.1 Data

To test the effects of translating data to English two types of data were used. One which had all formatting issues fixed, translating into English, removing random characters and stop words, lemmatisation and stemming applied. The other skipped the translating into English step. This formed the basis of either pipeline.

As also predicted in the data analysis section, the link(s), emoji(s) and username(s) columns were not added back to the data used in any models, as they all have negative effects on the models. Surprisingly, despite there being a higher chance of having fake text with emojis, adding the emojis back negatively affected both pipelines, the results of which can be seen in (Appendix G).

2.2.2 Feature Extractors

Two feature extractors were tested to see the effectiveness of either model. Count vectorizier and TFIDF vectorizer.

A count vectorizer works by creating a document of all words that appear in the dataset and for each tweet in the dataset, created a frequency vector from that document. For example, if a dataset consists of two tweets: "#hurricane #help #hurricane #GodHelpUs" and "#hurricane #sandy #NY #NewYork", after preprocessing, they would be represented as

hurricane	help	GodHelpUs	sandy	NY	NewYork
2	1	1	0	0	0
1	0	0	1	1	1

The advantage of this is that it can be easy to calculate, but it comes at a cost. Compared to other algorithms, it is unable to identify more or less important words and is unable to identify relationships between words. They also tend to give worse results than a TFIDF vectorizer.[1]

A Term Frequency-Inverse Document Frequency Vectorizer (TFIDFVectorizer) works similarly to a CountVectorizer, where it creates a document of all words, but instead of only calculating the frequency, it applies an importance cost to each word. For a term i in a document j each word is calculated using:

$W_{i,j} = tf_{i,j} \times \log(N / df_i)$ where:

- $tf_{i,j}$ = The number of occurrences of i in j
- df_i = The number of documents containing i
- N = The total number of documents
- $W_{i,j}$ = The cost of a word i in j

This penalises words that commonly appear in many texts (for example stop words) but favours those whose appear less by giving it a higher cost. This usually gives better results for most models, but they come at a cost of computational complexity, where they can take longer to calculate.[1]

For this, then both pipelines used a TFIDF Vectorizer as they should gain better results overall.

2.2.3 Machine Learning Models

After researching different models, four were initially picked to see their effectiveness on either dataset, with two being chosen from the best. These models included: Multinomial Naive Bayes, Passive Aggressive, Random Forest and Logistic Regression. After testing different hyperparameters, such as iterations, alpha values, and estimators, two were chosen; Multinomial

Naive Bayes and Random Forest Classifier. The results of which can be accessed in (Appendix F). Furthermore, the Passive Aggressive and logistic regression classifiers tended to give results around 81-85% accuracy, whilst sometimes going down to 50% despite using the same parameters, which made it unreliable at times to use and test.

Multinomial Naive Bayes was one of the most commonly used in many natural language processing/text classification problems when researching. It is based upon Bayes' Theorem of P(A|B) = P(A) * P(B|A)/P(B). As we use a feature extractor to create a "Bag of Words" each tweet is calculated to have a conditional probability of belonging to each class. This is done by calculating the occurrence of each feature belonging to that class and normalising the result [2]. It was primarily chosen as the classifier tends to also be quite simple and efficient in text classification compared to other algorithms[3]. This made it much easier to test and analyse.

The Random Forest classifier however is based upon a decision tree. It works by taking a subset of data points and features for constructing multiple decision trees. Then each decision tree an output is generated for which either by voting or averaging, a classification is given[4]. The downside of this is that it is highly more complex compared to the MultinomialNB classifier, making it harder to test. However, they tend to work well in regards to not overfitting the data and usually obtain higher accuracy[5].

3. Evaluation

Each pipeline was tested to maximise the accuracy and F1 score. We take the F1 score to be calculated as

F1 = (2 x Precision x Recall) / (Precision + Recall)
Where:
- Precision = TP/ (TP + FP)
- Recall = TP / (TP+FN)

As we are trying to make a classifier to identify fake text, fake should be set as positive, and real should be set as negative.

3.1 Pipeline I

Pipeline I consisted of using all text translated into English, having links, emojis and random characters removed, with both lemmatisation and stemming being applied and stop words being removed. The feature extractor used was a TFIDF vectorizer, where the maximum DF value was tested between 0-1 with an increment of 0.1. The model used was a multinomialNB classifier with an alpha between 0-3 with an increment of 0.01 and a threshold accuracy of 0-0.5 with an increment of 0.01.

The best result was achieved with the following values, with the confusion matrix found below:

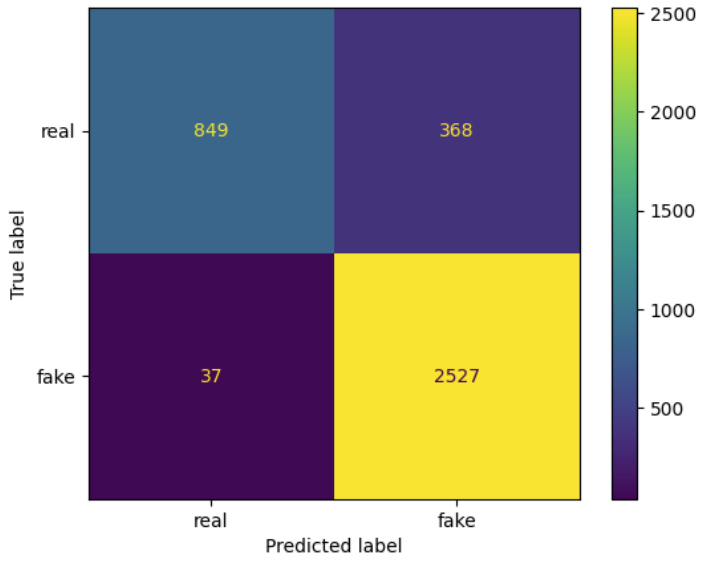

Figure 3.1.1. Confusion Matrix of MultinomailNB classifier with Alpha: 0.47 DF value: 0.10 Probability Threshold: 0.23 Accuracy: 89.29%

The F1 score was then calculated as to be:

- Precision = 2527/2527+368
- Recall = 2527/2527+37
- F1 Score = 0.9258

3.1 Pipeline II

Pipeline II consisting of using untranslated text. This was done as the Googletrans library rate limits the number of translations done per day, meaning if you were to use pipeline 1, you would likely be limited by how much data you could process per day. Links, emojis and random characters were still removed. Lemmatisation, and stemming were both applied to slightly lower the dimensionality of the dataset, as well as stop words being removed. This would only work for English languages, and languages which have words which would be detected by these libraries could incidentally be removed which could negatively impact the model. Similarly, the feature extractor used was a TFIDF vectorizer, where the maximum DF value was tested between 0-1 with an increment of 0.1. The model used was a Random Forest classifier with an estimator between 0-100 with an increment of 50 and a threshold accuracy of 0-0.5 with an increment of 0.05.

The best result was achieved with the following values, with the confusion matrix found below:

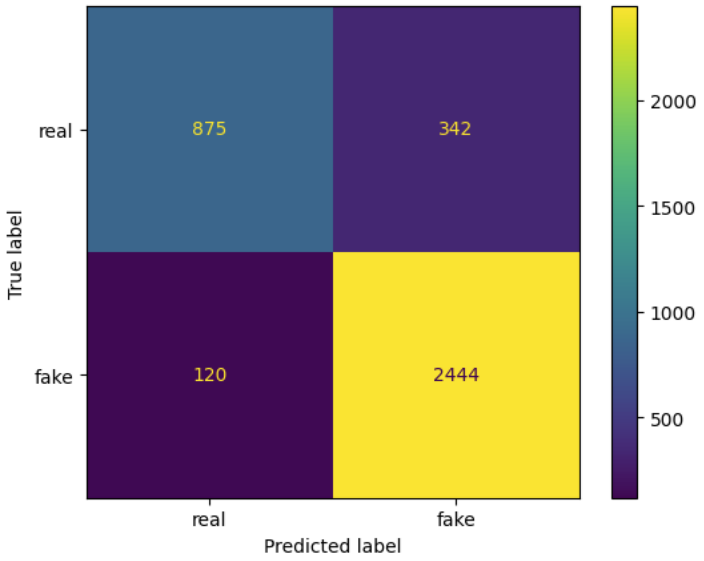

Figure 3.1.2. Confusion Matrix of Random Forest Classifier with Estimator: 50 DF value: 0.10 Probability Threshold: 0.05 Accuracy: 87.78%

The F1 score was then calculated as to be:

- Precision = 2444/2444+342
- Recall = 2444/2444+120
- F1 Score = 0.9136

The accuracies and F1 score for this however can vary as running it multiple times gives different outputs, even with the same values applied.

4. Conclusion

Pipeline I performed much better than Pipeline II in both the F1 score achieved as well as the run time being much lower. This was likely due to the translation of all the text to English, which did positively affect most models' accuracies. Pipeline II also suffered from giving different results despite the same parameters being used, which makes it less reliable overall compared to Pipeline I. The highest accuracy was used for Pipeline II, but it should be noted some results did range from 87-81%, sometimes reaching down to 50%. Having more estimators could solve this problem, but it does come at the cost of run time.

It should also be noted, that both pipelines were trying to maximise the accuracy and f1 score for the test dataset given. This likely could cause overfitting and may give lower accuracies with the same parameters if used on other datasets.

Some improvements could have been made to translating the text over multiple iterations to ensure all text was properly translated. Some papers also mentioned other feature extractors which worked better than the TFIDFVectorizer, like word2vec, which I would have liked to explore given more time[6]. Finding a better way to encode emojis may also help with improving the accuracy. Looking into spell checkers and dimensionality reduction could also help, but were not implemented due to time restrictions.

References

[1] Saket, S. (2022) Count Vectorizer vs TFIDF vectorizer: Natural language processing, LinkedIn. Available at: https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket/ (Accessed: 07 January 2024).

[2] Gomede, E. (2023) Understanding multinomial naive Bayes classifier, Medium. Available at: https://medium.com/@evertongomede/understanding-multinomial-naive-bayes-classifier-fdbd41b405bf (Accessed: 07 January 2024).

[3] Sriram (2022) Multinomial Naive Bayes Explained: Function, Advantages & Disadvantages, Applications in 2024, upgrad. Available at: https://www.upgrad.com/blog/multinomial-naive-bayes-explained/ (Accessed: 07 January 2024).

[4] R, S.E. (2024) Understand random forest algorithms with examples, Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/ (Accessed: 07 January 2024).

[5] Great Learning Team (2023) *Random Forest algorithm in Machine Learning: An overview*, *mygreatlearning*. Available at: https://www.mygreatlearning.com/blog/random-forest-algorithm/ (Accessed: 07 January 2024).

[6] Zhang, S., Wang, Y., & Tan, C. (2018). Research on Text Classification for Identifying Fake News. 2018 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC), 178–181. https://doi.org/10.1109/SPAC46244.2018.8965536 (Accessed: 07 January 2024).

Appendix

Appendix A: Longest Tweet in Dataset

"This can't be real...RT @ropgrady: \\355\\240\\275\\355\\270\\263\\355\\240\\275\\355\\270\\263\\355\\240\\275\\355\\270\\263 #wtf #hurricane #sandy #nyc #statueofliberty #waves \\355\\240\\274\\355\\274\\212\\355\\240\\275\\355\\267\\275⚡☔ @ NYC http://t.co/CiWRW36n"

Appendix B: Most Emojis Tweet in Dataset

'Man sandy Foreal??⚡⚡⚡☔☔⚡🌊🌊☁️🚣⛵💡🔌🚬🚬🚬🔫🔫🔒🔒🔐🔑🔒🚪🚪🚪🔨🔨🔨🏊🏊🏊🏊🎣🎣🎣😱😰😖😫😩😤💨💨💨💨💦💦💦💧💦💥💥💥👽💩🙌🙌🙌🙌🙌🙌🏃🏃🏃🏃🏃👫👫👩‍❤️‍👩👩‍❤️‍👨👩‍👩‍👧‍👧👩‍👩‍👦‍👦💑🙇🌕🌕🌕🌕🌎 http://t.co/vEWVXy10'

Appendix C: List of all events from the imageId(s) column with occurances

Training dataset:

- sandyA/B (Hurricane Sandy), 12523
- boston (Boston Marathon Bombings), 546
- malaysia (Flight MH370 Disappearance), 501
- sochi (Sochi Olympic Games), 402
- columbianChemicals (Chemical Plant Explosion), 185
- bringback (Girls kidnapped in Nigeria), 131
- underwater (An underwater Hotel bedroom), 113
- passport (A man being stuck in an airport after their kid ruined their passport), 46
- pigFish (A pig-like fish found in Brazil), 14
- elephant (An elephant carved out of rock), 13
- livr (A social media app you can only access whilst drunk), 9

Testing dataset:

- syrianboy (A syrian boy who rescued a girl in a shootout), 1786
- nepal (A historic building collapsing in an earthquake), 1360
- eclipse (A solar eclipse from the ISS), 277
- samurai (A samurai ghost found in a photo), 218
- garissa (A University Shooting killing 147 students in Garissa), 79
- varoufakis (A satirical video showing Varoufakis giving the middle finger), 61

Appendix D: Languages Detected in both Training and Test Dataset (results may vary once run)

en	13953	af	73	vi	14	hu	6
es	1360	ru	63	et	13	fa	5
so	629	sv	52	sq	11	hr	5
tl	304	tr	46	sw	10	lt	4
ar	270	ca	42	zh-cn	10	cs	2
fr	250	no	40	sl	8	te	2
pt	194	pl	39	bg	8	he	1
id	181	da	30	ko	8	not detected	1
de	169	ja	26	ro	8	mk	1
cy	126	th	20	el	7	lv	1
it	121	fi	17	hi	6	ta	1
nl	112	sk	15				

Appendix E: Languages detected using Real and Fake labels

Real Languages out of 6226

{'en': 5307, 'es': 305, 'fr': 105, 'tl': 96, 'de': 63, 'so': 48, 'it': 40, 'cy': 29, 'ar': 28, 'sv': 26, 'pt': 18, 'af': 17, 'nl': 17, 'id': 17, 'pl': 15, 'ru': 15, 'no': 13, 'tr': 10, 'da': 9, 'ca': 8, 'sq': 6, 'sk': 5, 'th': 3, 'ja': 3, 'et': 3, 'lt': 3, 'ro': 3, 'fi': 2, 'vi': 2, 'el': 2, 'hi': 2, 'te': 2, 'bg': 1, 'mk': 1, 'fa': 1, 'sl': 1}

Fake Languages out of 12038

{'en': 8641, 'es': 1057, 'so': 597, 'ar': 239, 'tl': 217, 'pt': 179, 'id': 160, 'fr': 143, 'de': 103, 'nl': 100, 'cy': 97, 'it': 78, 'af': 59, 'ru': 48, 'tr': 36, 'ca': 33, 'no': 23, 'ja': 22, 'sv': 22, 'pl': 20, 'da': 18, 'th': 18, 'fi': 17, 'vi': 13, 'sk': 11, 'sw': 11, 'zh-cn': 10, 'bg': 8, 'ko': 7, 'hu': 6, 'sl': 6, 'sq': 5, 'hr': 5, 'el': 5, 'et': 5, 'lt': 4, 'ro': 4, 'hi': 4, 'cs': 2, 'he': 1, 'fa': 1, 'lv': 1, 'ta': 1}

Appendix F: Best Results of Models using a TFIDFVectorizer Feature Extractor:

MultinomialNB Classifier with Translated Text:
Best Accuracy: 0.89288548 Alpha: 0.47 DF value: 0.10 Probability Threshold: 0.23

Passive Aggressive Classifier with Translated Text:
Best Accuracy: 0.81645067 Iterations: 330 DF value: 0.70 Probability Threshold: 0.01

Logistic Regression Classifier with Translated Text:
Best Accuracy: 0.84765935 DF value: 0.30 Probability Threshold: 0.04

RandomForestClassifier without Translating Text:
Best Accuracy: 0.87781010 Estims: 50 Df value: 0.10 Probability Threshold: 0.05

Appendix G: Pipelines using Emojis Column:

Pipeline I with Emojis:
Best Accuracy: 0.89050516 Alpha: 0.38 DF value: 0.10 Probability Threshold: 0.22

Pipeline II with Emojis:
Best Accuracy: 0.85268447 Estims: 50 Df value: 0.10 Probability Threshold: 0.05